Hyperledger Fabric

1) What is Hyperledger Fabric ?

Hyperledger Fabric is one of the blockchain projects within Hyperledger. Like other blockchain technologies, it has a ledger, uses smart contracts, and is a system by which participants manage their transactions.

Hyperledger Fabric — hosted under Linux Foundation — is a private, permissioned and open source blockchain solution. Private means that blockchain networks are not publicly accessible and only invited parties can join the network. Permissioned means each party is clearly identified and every transaction is authenticated, authorized, validated and tracked. You can run Fabric networks on-premise or use Blockchain as a Service platforms to maintain the ledger infrastructure for you.

2) Organization

An organization in Hyperledger Fabric corresponds to a real organization, company, political party, etc. Each organization in Fabric has to dispose of a Public Key Infrastructure (PKI) which has to be comprised of a Certificate Authority (CA) and possibly even an Intermediate Certificate Authority (ICA) that will issue digital certificates for the organizations identities (e.g. users, client applications, peers, orderers). Those then use them to authenticate themselves in the messages they exchange within the networks.

Hyperledger Fabric comes with a default CA service but it is not necessary to use the default service for your PKI management. You may use any already existing PKI that your organization has. Another option is to utilize HashiCorp Vault or other solutions to issue public-private key pairs for your organization. The only important thing is to create the keys and certificates in the supported format — signing keys must be in the ECDSA format. The RSA format is not currently supported.

Membership Service Provider (MSP)

Membership Service Provider (**MSP**) is a component that aims to offer an abstraction of a **membership** operation architecture. In particular, **MSP** abstracts away all cryptographic mechanisms and protocols behind issuing and validating certificates, and user authentication. As mentioned in MSP description, MSPs may be configured with a

set of root certificate authorities (rCAs), and optionally a set of intermediate certificate authorities (iCAs).

In Fabric, an organization is identified by its MSP. In Hyperledger Fabric Network each organization is identified by its Membership Service Provider identification (MSP ID). To be accurate, an organization can be comprised of one or multiple MSPs but for most cases, and for the sake of simplicity, you will probably use a single MSP to represent an organization. An MSP is formed by **an instance of a PKI infrastructure** which is then able to issue certificates for the MSP.

3) Peers

A peer is a node running on the Hyperledger Fabric peer binary. Each organization is supposed to have peers to host ledgers and smart contracts. Every channel (network) has its own data in a separate ledger stored on peers. And every channel is supposed to have one or more smart contracts (chaincodes). Multiple versions of a chaincode can be installed and stored on organizational peers and can be instantiated into one or multiple channels. Applications connect to peers to query (get) or invoke (put) data into ledgers. For query, only peers are contacted.

There are different types of peer nodes with different roles in the network:

- Endorser peer
- Anchor peer
- Orderer peer

Endorser peer

Peers can be marked as Endorser peer (ie Endorsing peer). Upon receiving the “transaction invocation request” from the Client application the Endorser peer

- Validates the transaction. ie Check certificate details and roles of the requester.
- Executes the Chaincode(ie Smart Contract) and simulates the outcome of the transaction. But it does not update the ledger.

At the end of the above two tasks, the Endorser may approve or disapprove the transaction.

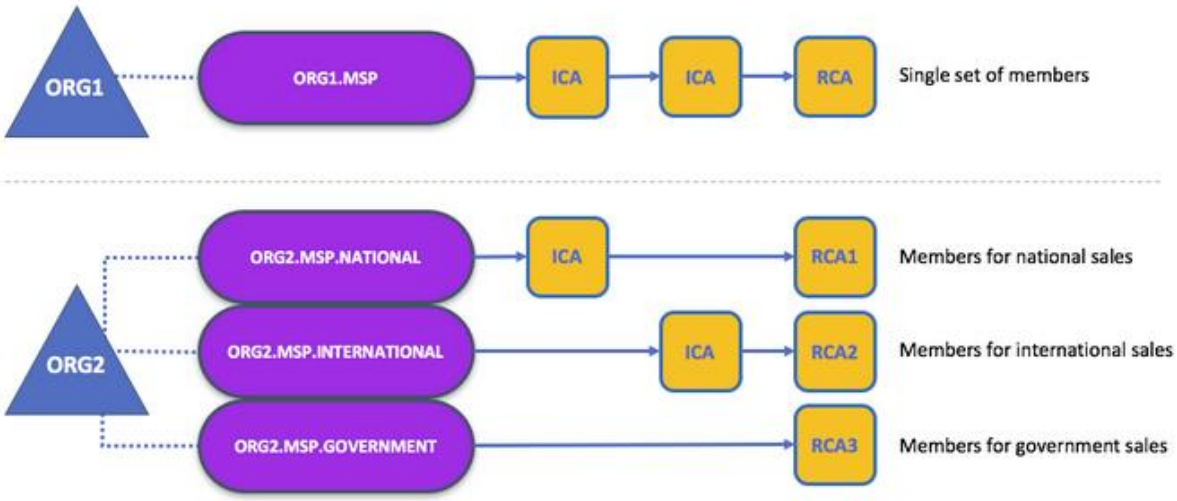
As only the Endorser node executes the Chaincode (Smart Contract) so there is no necessity to install Chaincode in each and every node of the network which increases the scalability of the network.

Anchor Peer

A peer node on a channel that all other peers can discover and communicate with. Each member on a channel has an anchor peer (or multiple anchor peers to prevent single point of failure), allowing for peers belonging to different Members to discover all existing peers on a channel.

Anchor peer or cluster of Anchor peers is configured at the time of Channel configuration. Just to remind you, in Hyperledger Fabric you can configure secret channels among the peers and transactions among the peers of that channel are visible only to them.

Anchor peer receives updates and broadcasts the updates to the other peers in the organization. Anchor peers are discoverable. So any peer marked as Anchor peer can be discovered by the Orderer peer or any other peer.



Hyperledger Fabric Membership Service Provider (MSP) hierarchy diagram (image source)

Consensus

In a distributed ledger system, **consensus** is the process of reaching agreement on the next set of transactions to be added to the ledger. In Hyperledger Fabric, consensus is made up of three distinct steps:

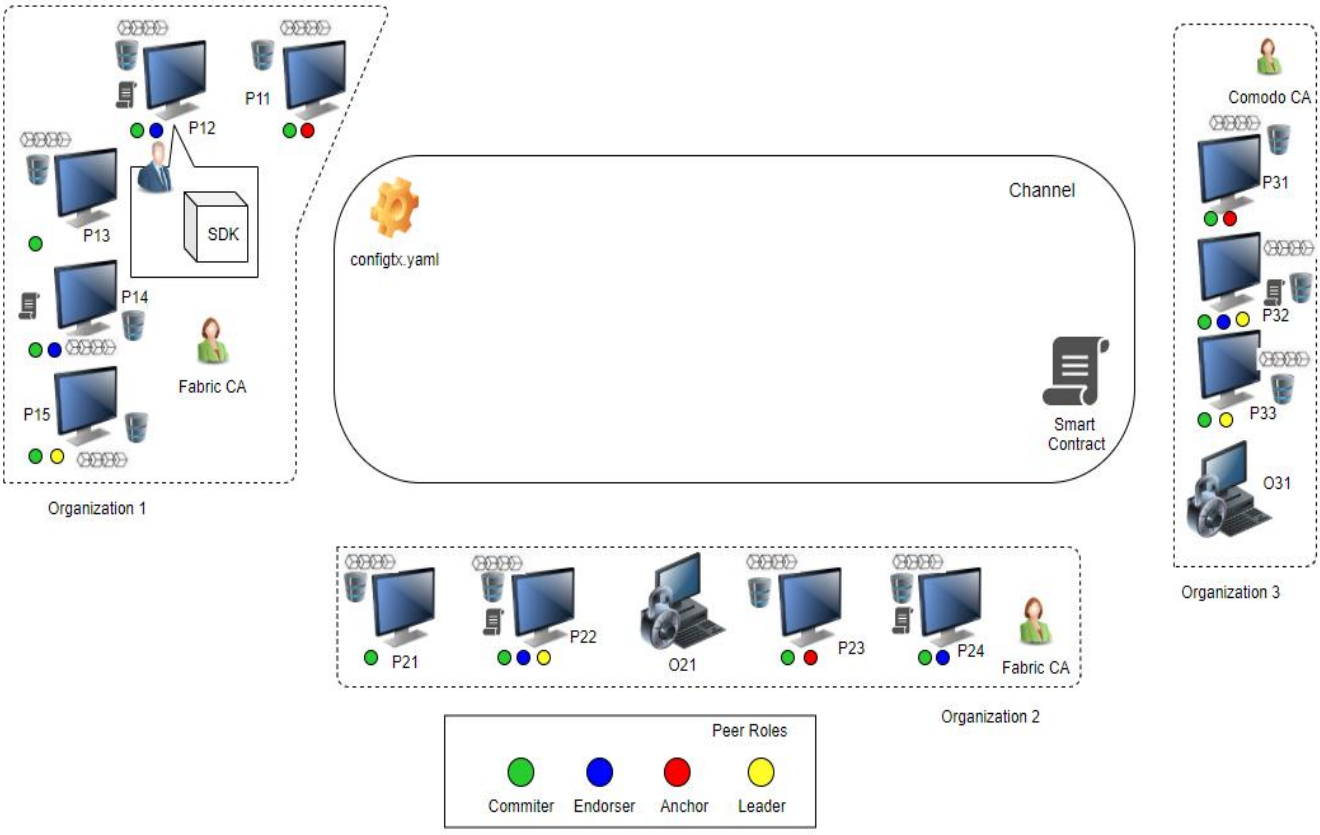
- Transaction endorsement
- Ordering
- Validation and commitment.

Hyperledger Fabric SDKs

Hyperledger Fabric intends to offer a number of SDKs for a wide variety of programming languages. The first two delivered are the Node.js and Java SDKs. We hope to provide Python, REST and Go SDKs in a subsequent release.

4) Network Setup

To start with, first, we'll consider a full-fledged fabric network, production like. Let's go through the network setup.



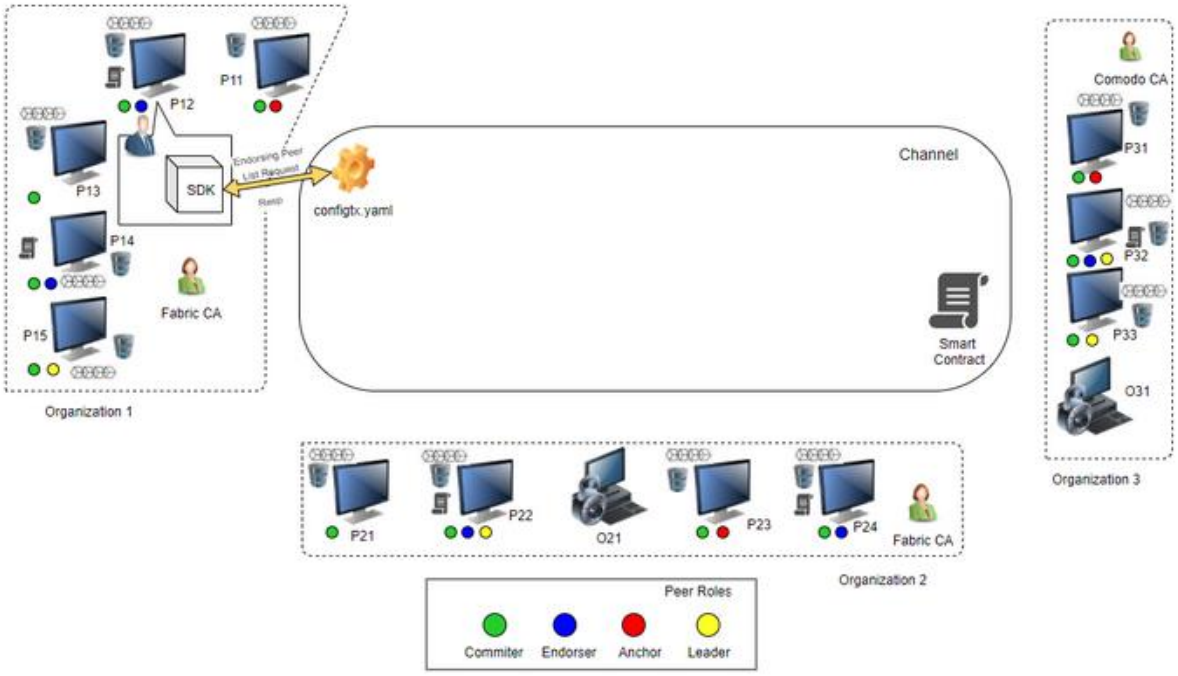
Consider a Hyperledger Fabric Multi organization network as shown above. This network is formed by three organizations. Org1 which has dedicated 5 peers, Org 2 with 4 peers and 1 orderer and Org 3 with 3 peers and 1 Orderer. We have channels formed by these organizations. Any transaction carried out by any of the organization is transparent to all 3 organizations. A channel has an instance of Smart Contract and channel policy which are defined in configtx.yaml, which we pass as an input during the creation of the channel.

By default, each peer has a role of Committer. In addition to this, a peer can possess multiple roles like Endorser, Leader or Anchor. All Peers are represented by a colored dot representing the set of roles a peer possess. A single peer can be set with all roles. Each peer possesses a

ledger which consists of a blockchain and a World State. All peers who possess endorsing role has Smart contract installed in order to validate the transactions.

Each organization has its own CA authority. Since Fabric is a modular structure, we can have any CA authority like Comodo, DigiSign, etc. here in our network Organization 1 and 2 associated with Fabric CA and Organization 3 associated with Comodo CA.

Step 1: Client Initiates the transaction

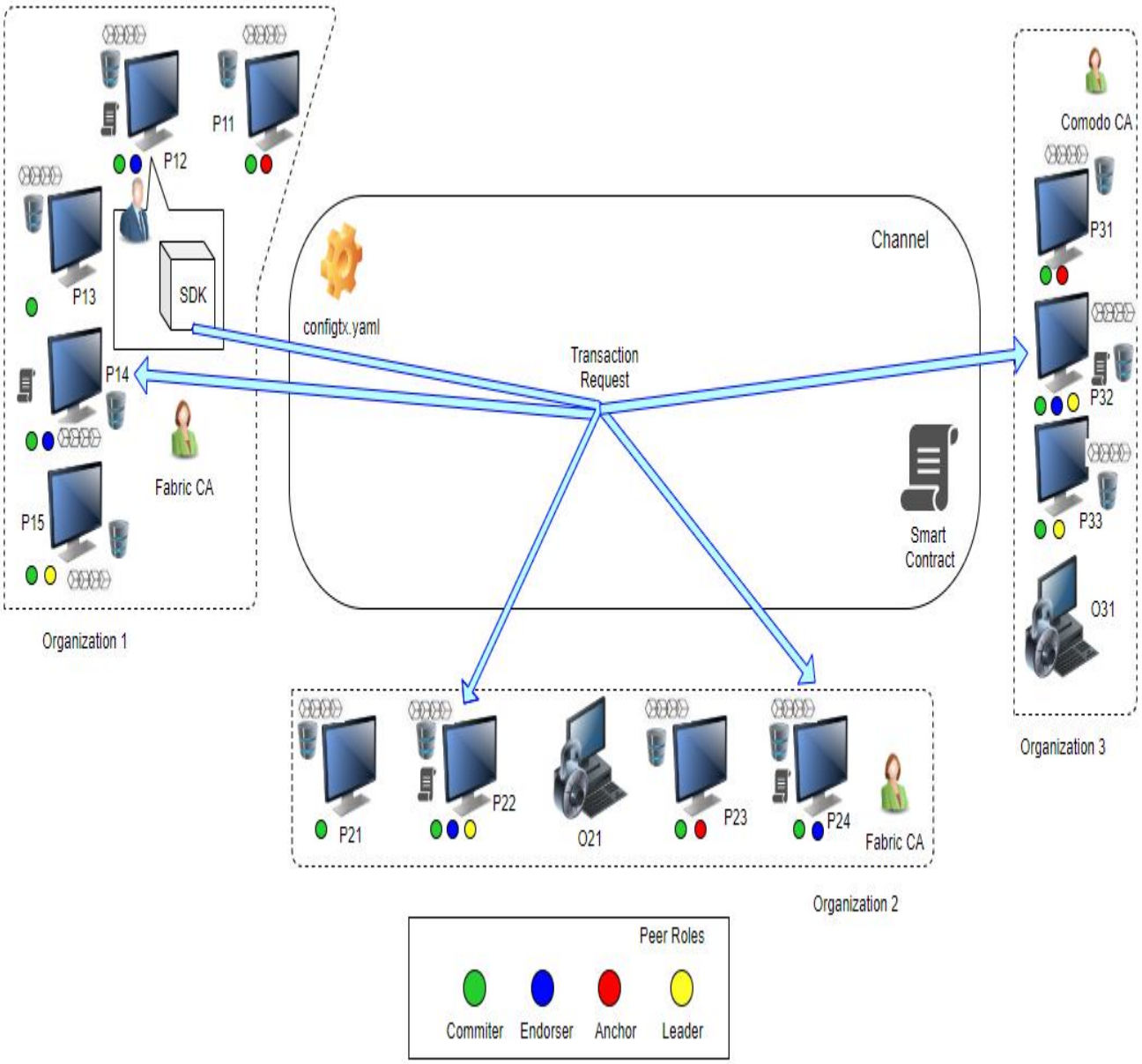


Client requests for Endorsement lists

Client requests for Endorsement lists

Within a Hyperledger Fabric network, transactions carried out by a client application, sending transaction proposal, or, in other words, proposing a transaction to endorsing peers. Before sending the transaction to endorsing peer, client SDK needs to know the list of all Endorsing peers in the network. Here in our already defined network, we have 5 endorsing peers set as a blue dot. These endorsing peers are defined in configtx.yaml file and is set as an endorsement policy while creating it. Each endorsing peer has a chaincode installed and has a copy of ledger which is sync between all peers. Along with the list of endorsing peers, requesting peer learns about the rule for a number of endorsers required to achieve consensuses. Unless this rule is satisfied, the requesting peer does not send this message further to other peers. For Example, consider our network, where policy specifies that “at least one endorsing peer from Org 1 and Org 3 or all endorsing peers from Org2 and Org3”. Until this policy is not satisfied, requesting peer will not be able to send a message to other endorsing peer.

Step 2 — Endorsing peers verify signature & execute the transaction



Transaction requests to all Endorsing peers

Using this SDK a transaction proposal is constructed, it uses an SDK API to generate this transaction proposal. This proposal is nothing but a request to invoke a chaincode, in order to read or write in the ledger. The SDK is nothing but a process to bundle a request into a proper package which is accepted by the network which takes the clients cryptographic credentials to generate a signature for this transaction proposal. To invoke a transaction, the client sends a "PROPOSE" message to a set of endorsing peers of its choice. Some endorsers could be offline, others may object and choose not to endorse the transaction.

Once the transaction request is formed, it is now sent to all endorsing peers on the list to get a consensus that the transaction is valid. Upon receiving the message it's a responsibility of each endorsing peer to follow predefined steps to validate the message as below.

Each endorsing peers will validate:

1. If the proposal is well-formed
2. Not been submitted in past
3. Signature is valid
4. The submitter(Client) is authorized to carry out the execution on the channel.

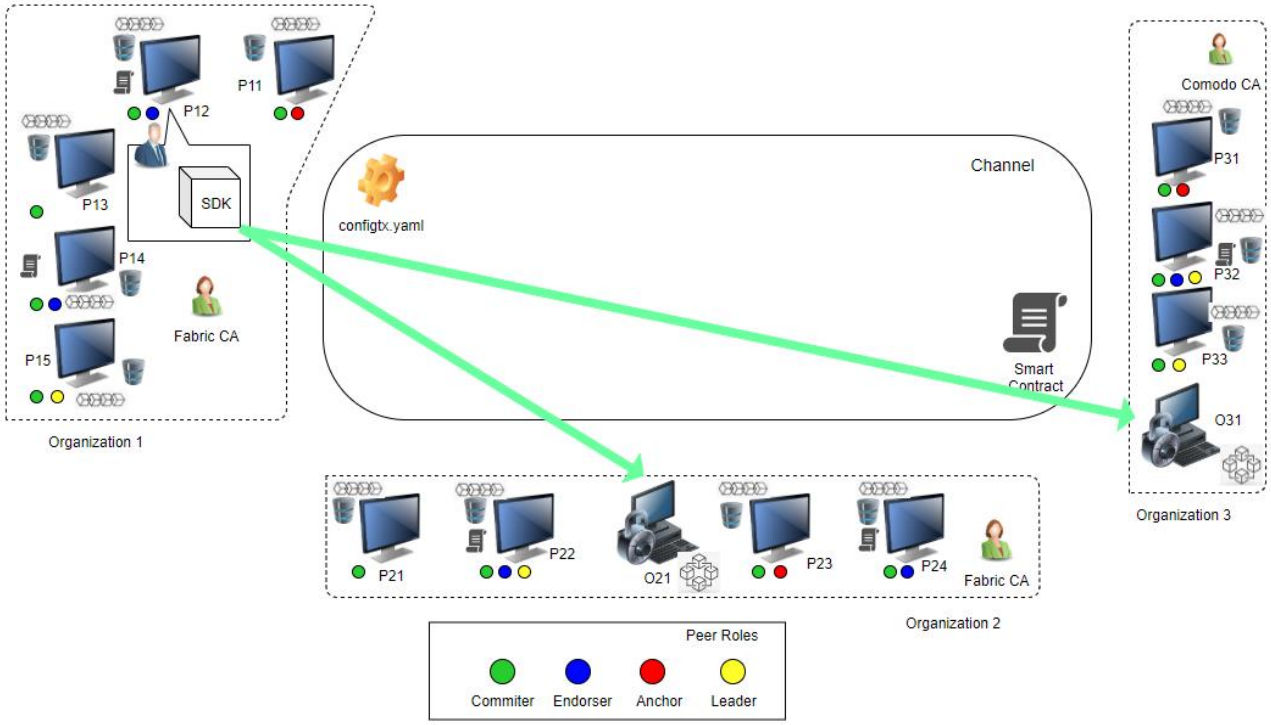
Each endorsing peer takes the transaction proposal inputs as arguments to invoke the chaincode function. Chaincode is then executed against the current state of the database to produce transaction results including response value, along with read and write sets, called RW Sets. These RW sets capture what was read from the current world state while simulating the transaction, as well as what would have been written to the world state had the transaction been executed. No updates are made in a ledger at this point. The set of these values, along with the endorsing peer's signature is passed back as a "proposal response" to the SDK which parses the payload for the application to consume. Now the requesting peer receives an Endorsement transaction response from all endorsing peer. It's a list of endorsement responses. After receiving the list of endorsements, this list is validated based on the rule set in the endorsement policy and checks if it is satisfied or not.

Since an endorsement policy is specified for a specific chaincode, different channels can have different endorsement policies.

Step 3 — Proposal response are inspected

The application verifies the endorsing peer signature and compares the proposal response to determine if the proposal responses are the same. If the chaincode only queried the ledger, the application would inspect the query response and would not submit the transaction to the Ordering service. If the client application intends to submit the transaction of the Ordering service, the application identifies if the specified endorsement policy has been fulfilled before submitting.

Step 4 — Client assembles endorsements into a transaction



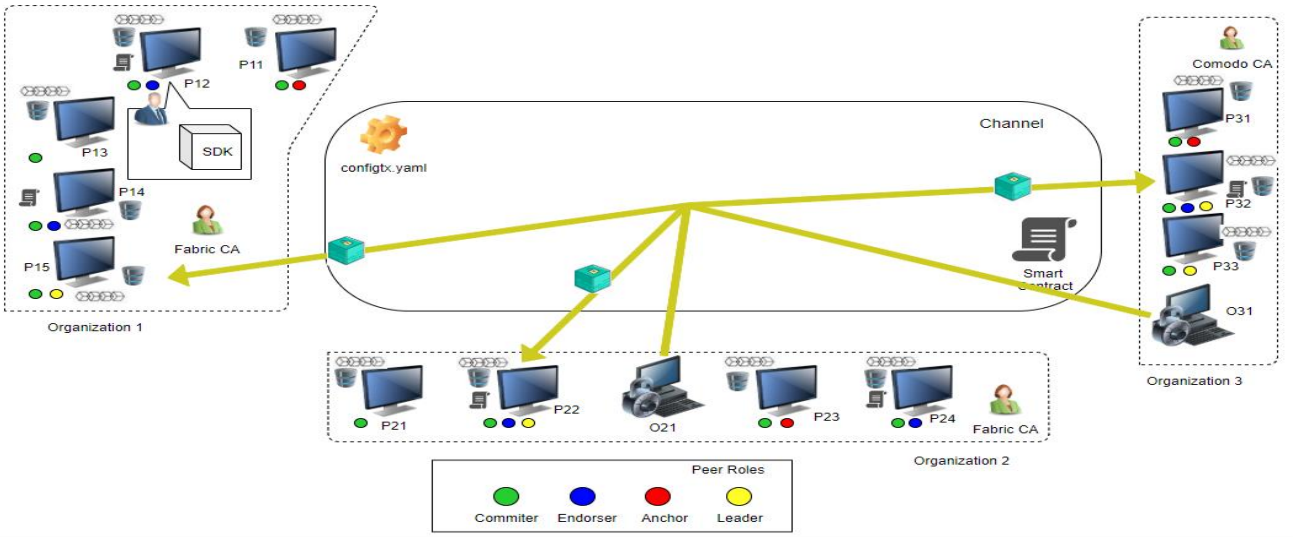
Endorsements being passed to orderers

Once all Endorsement responses are validated and found suitable to deliver it to the Ordering service, these messages then start there journey to form a block. In our current network, we have two ordering services from Org 2 and Org 3. Usually, in production, we should set up multiple ordering services to make sure the network is fault tolerant. However, we can use Kafka which enables fault tolerance and also provides high-throughput, low-latency for real-time data feeds.

The SDK broadcasts the transaction proposals and response within transaction message to the Ordering service. This message contains a transaction which contains read/write sets, endorsing peer signature and channel ID. The Ordering service does not inspect every information in this message, it simply receives all transactions from all channels in the network, orders them chronologically and creates a block of transactions per channel. Ordering happens across the network, in parallel with endorsed transactions and RW sets submitted by other applications.

The Ordering service, which is made up of a cluster of orderers, does not process transactions, smart contracts, or maintain the shared ledger. The ordering service accepts the endorsed transactions and specifies the order in which those transactions will be committed to the ledger. The Fabric v1.0 architecture has been designed such that the specific implementation of "ordering" (Solo, Kafka, BFT) becomes a pluggable component. The default ordering service for Hyperledger Fabric is Kafka. Therefore, the ordering service is a modular component of Hyperledger Fabric.

Step 5: Disseminate the block to leader peers

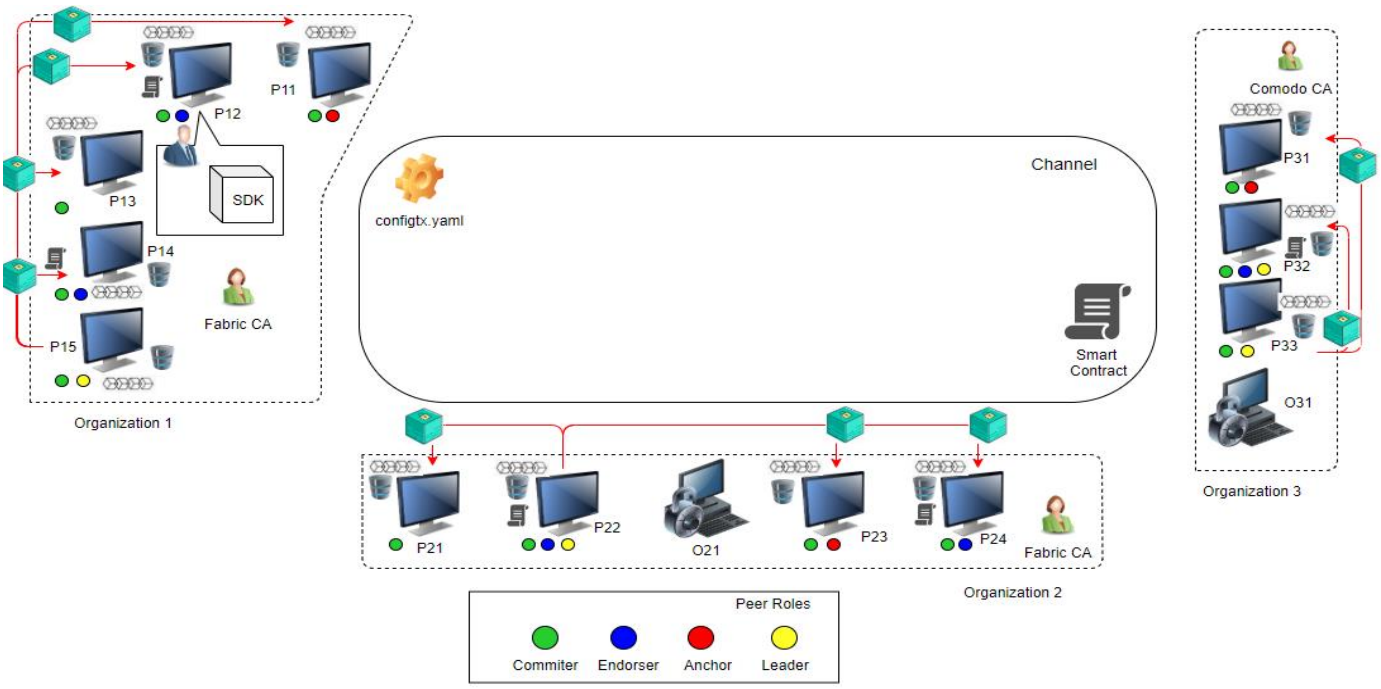


Block disseminate to leaders

Once the block is formed by arraigning the transactions in chronological order, the block is now ready to send to all leader peers in the network. Ideally, the network should have a leader peer, but it's not mandatory. If there are no leader peers the orderer will need to make sure that the block is propagated to all committing peers, which could cause unnecessary tracking and network load. To avoid this, each organization nominates there own leader peer. There can be multiple leader peer per organization, in order to be fault tolerant. If an organization has only one leader peer and fails to respond, a voting mechanism takes place to nominate the leader peer from available active peers. The list of leader peers is known to all orderers via the channel.

The orderer sends this message block to all active leader peers.

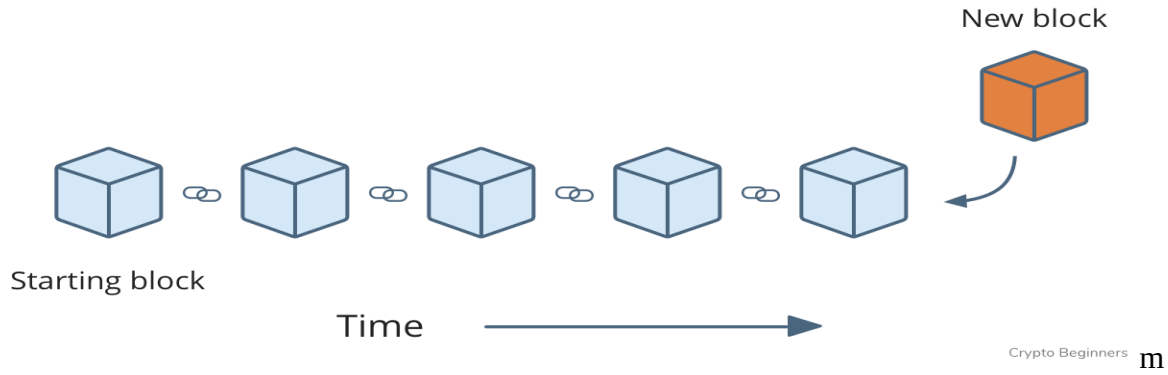
Step 6: Transaction is validated and committed



Blocks delivered to all Committing peers.

Leader peers come only into the picture at this stage, to disseminate the block to other committing peers within the organization. The block is delivered to all peers on the channel using gossip protocol. The transactions within the block are validated to ensure endorsement policy is fulfilled and to ensure that there have been no changes to ledger state for read set variables since the read set was generated by the transaction execution. Transactions in the block are tagged as being valid or invalid.

Step 7 — Ledger update



Peers checks if the endorsement is valid according to the policy of the chaincode and also verifies the dependencies (RW sets) are not been violated. The committing peer validates the transaction by checking to make sure that the RW sets still match the current world state.

Specifically, that the Read data that existed when the endorsers simulated the transaction is identical to the current world state. When the committing peer validates the transaction, the transaction is written to the ledger, and the world state is updated with the Write data from the RW Set. Each peer appends the block to the channel's chain, and for each valid transaction, the write sets are committed to the current state database. All valid transactions are marked as valid using setting the bitmask value as 1 and 0 for invalid.

Each peer possesses a single blockchain per channel and a peer can be associated with multiple channels, hence can have multiple blockchains.

Step 8 — Client notification

Lastly, the committing peers asynchronously notify the client application of the success or failure of the transaction. Applications will be notified by each committing peer.

These are the detailed steps of Fabric Transaction flow. I tried to cover as much as in detail. Hope you liked the technical article. Any feedback and suggestions are most welcome.

5) Overview of our layered design

The Hyperledger Fabric platform consists of four parts: Blockchain member, ordering service, endorsing peers and committing peers.

- Blockchain member: These devices are collecting and transmitting data to private Blockchain distributed ledgers.

- Endorsing peers: During the commissioning and configuration of the Blockchain network, the developers should select a number of connected devices are defined as the endorsing peers.

- Ordering service: creates the block of transactions, and sends it to all the peers. The ordering service collects transactions for a channel into proposed blocks for distribution to peers. Blocks are delivred on a channel basis. The ordering service accepts endorsed transactions, orders them into a block, and delivers the blocks to the committing peers. The main responsibility of ordering service is to receive transactions from the Blockchain devices and fit into a Block.

Committing peers: A committing peers are a predefined number of Blockchain devices. Usually endorsing peers are also committing, but a peer can be only committing and not endorsing. Committing peers (including endorsing peers) run validation and update their copy of the Blockchain and world state. Each peer receiving the block, now in the role of a committing peer, appends the whole block to its Blockchain copy. Committing peers are responsible for adding blocks of transactions to the shared ledger and updating the world state. They may hold smart contracts, but it is not a requirement.

1. The Blockchain device submits endorsement requests to endorsing peers.

2. Endorsing peers approve or reject the status after checking for consistency (checking the smart contract).

3. Endorsers peers send the proposal response to the IoT device applicant. Each execution captures the set of read and written data (also called the RW set), which is flowing in the Hyperledger fabric. Moreover, the Endorsement System Chaincode (ESCC) signs the proposal response on the endorsing peer . The RW sets are signed by each endorser. Every set will include the number of the version of each record.

4. Afterwards, the ordering service verifies the collected endorsements and creates the Block of transactions

5. The ordering service sends the created Block to committing peers of the Blockchain

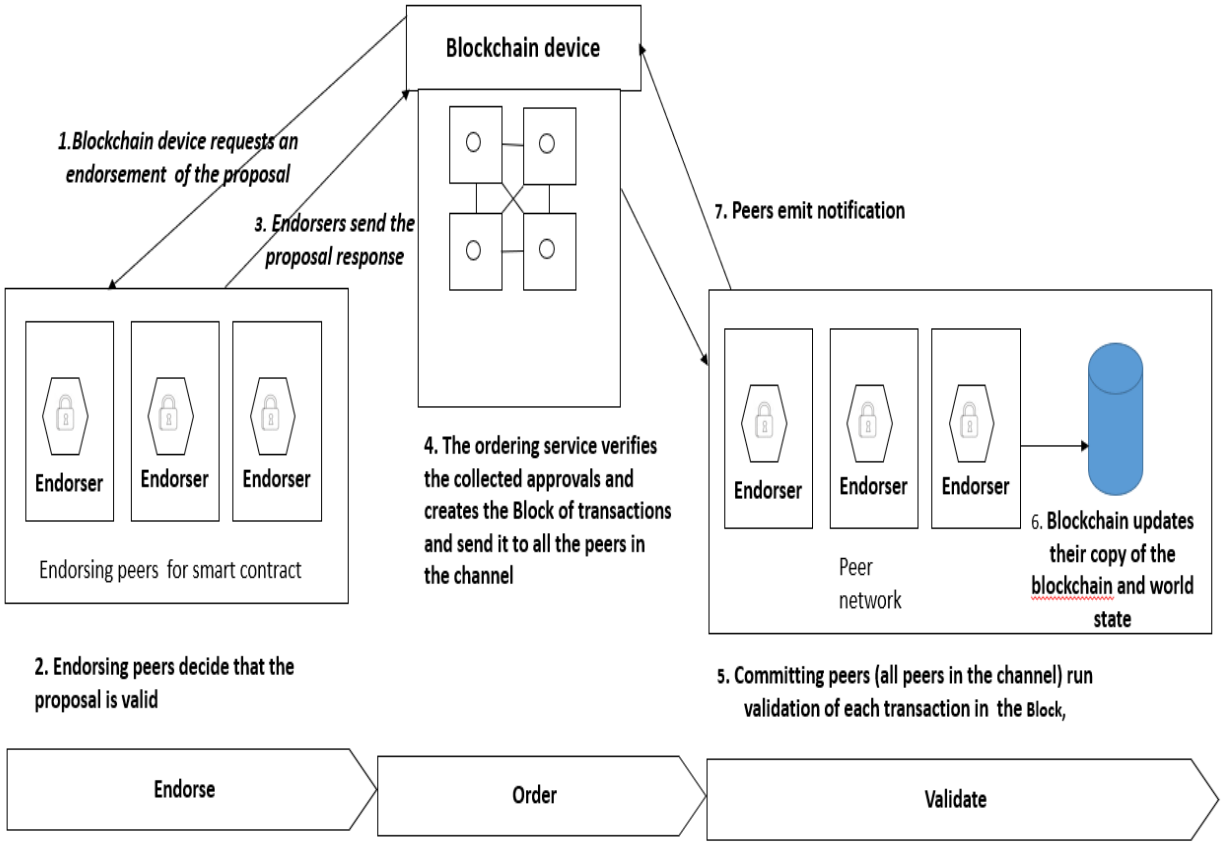
6. The committing peers updates the received Block and adds transactions received from different Blockchain devices.

7. Then, the Blockchain sends the Block to all the peers of the network to be appended to their Blockchain copies.

8. The peers check the transaction's validity (verification of the smart contract).

9. If the validity is fulfilled, finally the Blockchain platform adds the checked Block into the data base of the Blockchain platform.

10. Peers emit notification



6) Smart Contracts for monetizing Blockchain data

The concept of smart contracts can be used to automate negotiations between service providers and users along with required monetary transactions without a trusted intermediary. A Smart Contract is fundamentally a code that validates a negotiation and immediately brings a contract into effect, without the involvement of any intermediaries. The smart contract code resides on a blockchain as multiple functions with unique addresses that can be called by any user of the Blockchain. All entities interacting with a Blockchain (including users and devices) must own at least one public-private key pair. First of all the sender encrypts a smart contract with the public key. The receiver then receives the encrypted smart contract and decrypts it with the private key.

In Hyperledger Fabric terms, a smart contract is a piece of code (written in Go, node.js, or Java) enforcing contract processing between parties. Chaincode is a packaged smart contract code prepared for installation. Essentially, a smart contract lies within a chaincode package.

In general, the smart contract is where automation of processes between organizations comes into place. Smart contracts work with data on the blockchain to typically automate business as well as workflows. Most of smart contract automation is written in a way that if something happens it triggers a different action that will be automatically written to the ledger. An example of smart contract automation can be tracking an SLA, and if it is breached it will automatically commit a discount to the ledger for the provided IT service. A different example can be how within elections a smart contract automatically commits a winner on the ledger when all the votes are counted. Of course more actions or chain of actions can be implemented.